Algorithm 1 Propagate Light Distribution

1: **procedure** PROPAGATEDIST
2: FILL(*bufferA*, 0.0) ▷ Reset and Initialize
3: FILL(*bufferB*, 0.0)
4: $sumMem \leftarrow 0.0$
5: $finished \leftarrow false$
6: $index \leftarrow (j * width + i) * steps + t$ ▷ compute $1D$ index
7: $bufferA(index) \leftarrow steps$ ▷ write light src
8: **while** $\Delta\Phi_{total} < \epsilon$ **do** ▷ while convergence criterion not met..
9: $sumA \leftarrow 0.0$ ▷ reset sum
10: PROPAGATE() ▷ propagate *bufferA* (src) in *bufferB* (tar)
11: $sumA \leftarrow$SUM(*bufferB*) ▷ sum up energies
12: $\Delta\Phi_{total} \leftarrow |sumA - sumMem|$ ▷ compute difference to prev. iter.
13: $sumMem \leftarrow sumA$ ▷ save sum for next iter.
14: SWAP(*bufferA*, *bufferB*) ▷ swap buffers for restart
15: $bufferA(index) \leftarrow steps$ ▷ re-write light src
16: FILL(*bufferB*, 0.0)
17: **if** $ctr > limit$ **then** ▷ stop on iteration limit
18: break
19: **end if**
20: **end while** ▷ final light distribution stored in bufferA..
21: return *bufferA*
22: **end procedure**

Algorithm 2 Propagate BufferA to BufferB

procedure PROPAGATE
  for each cell c in bufferA **do**
    $\mu_{I_c} \leftarrow mean(I_c)$ ▷ Compute Mean Intensity
    if $\mu_{I_c} = 0.0$ **then** ▷ break on trivial null sample
      break
    end if
    $\mu_{T_c} \leftarrow mean(T_c)$ ▷ Comp. Mean Transmission
    $\mu_{T_c I_c} \leftarrow mean(T_c \cdot I_c)$ ▷ Comp. Mean Transmitted Intensity
    $n_c \leftarrow \frac{\mu_{T_c}\mu_{I_c}}{\mu_{T_c I_c}}$ ▷ Comp. normalization factor for cur. cell
    for each direction k in $[0, 7]$ **do** ▷ For all neighbors, do..
      $\gamma \leftarrow k\frac{\pi}{4}$ ▷ Offset to compute cone center angle
      if $k \bmod 2 = 0$ **then** ▷ if even (face) neighbor..
        $energy = \int_{\gamma-\pi/4}^{\gamma+\pi/4} n_c \epsilon_\alpha T_c(\omega) I_c(\omega)\, d\omega$
      else ▷ if odd (diagonal) neighbor..
        $energy = \int_{\gamma-\beta}^{\gamma+\beta} n_c \epsilon_\beta T_c(\omega) I_c(\omega)\, d\omega$
      end if
      $index \leftarrow c + deltaIndex(k)$ ▷ assign neighbor destination index
      $bufferB(index) \leftarrow energy \cdot cos_k(\omega) + bufferB(index)$ ▷ Comp. y=a*x+y
        ▷ Scale cosine lobe with summed energy and accumulate → daxpy-OP
    end for
  end for
end procedure